ECE232: Hardware Organization and Design

Lecture 17: Pipelining Wrapup
Outline

- The textbook includes lots of information
  - Focus on material covered on homeworks, lecture, and discussion

- Most important
  - Read after write hazards
  - Control hazards (branches)

- Today:
  - Wrapup important pipelining issues
  - Talk about improving control hazard performance
Forwarding

Instr. Order

add $1,...

sub $4,$1,$5

and $6,$7,$1

or $8,$1,$1

sw $4,4($1)
Data Forwarding (aka Bypassing)

- Take the result from the point that it exists in *any* of the pipeline state registers and forward it to the functional unit (e.g., the ALU) that needs it that cycle.

- For ALU functional unit: the inputs can come from *any* pipeline register rather than just from ID/EX by
  - **add multiplexors to the inputs of the ALU**
  - **connect the result data in EX/MEM or MEM/WB to both of the EX’s stage Rs and Rt ALU mux inputs**
  - **add the proper control hardware to control the new muxes**

- Other functional units may need similar forwarding logic (e.g., the DMem)
Datapath with Forwarding Hardware

Instruction Memory
Read Address
Add

Data Memory
Address
Read Data
Write Data

Control
IF/ID

Forward Unit

ALU cntrl
ALU
Shift left 2
Add

Branch
MEM/WB

PCSrc

Shift left 2
Add

PCSrc
Overall

- All modern day processors use pipelining
- Pipelining doesn’t help latency of single task, it helps throughput of entire workload
- Pipeline clock cycle determined/limited by slowest pipeline stage
  - Unbalanced pipe stages cause inefficiencies
  - The time to “fill” pipeline and time to “drain” it can impact speedup for deep pipelines and short code runs
- Must detect and resolve hazards
Review: Pipeline Hazards

- **Structural hazards**
  - Design pipeline to eliminate structural hazards

- **Data hazards** – read after write - RAW
  - Use data forwarding inside the pipeline
  - For those cases that forwarding won’t solve (e.g., load-use) include hazard hardware to insert stalls/bubbles

- **Control hazards** – beq, bne, j, jr, jal
  - Stall – hurts performance
  - Move decision point as early in the pipeline as possible – reduces number of stalls at the cost of additional hardware
  - Delay decision (requires compiler support) – “Delayed Branch”
  - Predict outcome of Branch
    - Static prediction – e.g., always not-taken
    - Dynamic prediction – prediction per branch in program
Branch Instructions Cause Control Hazards

beq  lw
Inst 3  Inst 4

jr

F  D  EX  M  W
BEQ resolved during the MEM stage
One Way to “Fix” a Control Hazard

Fix branch hazard by waiting – introduce stalls
Reducing Control Hazards’ Penalties

- Stalls – hurts performance
- Deeper pipelines have higher penalties
- 1. Move decision point as early in the pipeline as possible – reduces number of stalls at the cost of additional hardware
- 2. Delay decision (requires compiler support) – “Delayed Branch”:

  \[
  \begin{align*}
  &\text{beq } $1,$2,NEXT \\
  &\text{add } $4,$3,$5 \\
  &\text{sub } $7,$2,$8 \\
  &\ldots \\
  &\text{NEXT}
  \end{align*}
  \]

  - not effective for deeper pipes - requiring more than one delay slot to be filled
- 3. Predict outcome of branch
Reducing branch penalty through HW design
Branch Prediction

- Easiest - static prediction
  - Always taken, always not taken
  - Opcode based
  - Displacement based (forward not taken, backward taken)
  - Compiler directed (branch likely, branch not likely)

- Dynamic prediction – prediction per branch in program
  - **1 bit predictor** – remember last taken/not taken per branch
    - Use a *branch-history table (BHT) with 1 bit entry*
    - Use part of the PC (low-order bits) to index table – *Why?*
      - Multiple branches may share the same bit
    - Invert the bit if prediction is wrong

![BHT Diagram]

```plaintext
BHT

Predictor 0
Predictor 1

Branch PC

Predictor 127
```
Branch Prediction

- **1 bit predictor**
  - Backward branches for loops will be mispredicted twice
    - EX: If a loop branches 9 times in a row and not taken once, what is the prediction accuracy?
      - Misprediction at the first and last loop iteration => 80% prediction accuracy, although branch is taken 90%

- Modern processors – multiple instructions issued per cycle, more branch hazards will occur per cycle
  - Cost of branch mispredicted goes up
  - Pentium II – 3 instructions issued per cycle, 12+ cycle misprediction penalty
    - Huge penalty for a misfetched path following a branch
2-bit Branch Prediction

- 4 states instead of 2, allowing for more information about tendencies
- A prediction must miss twice before it is changed
- Good for backward branches of loops
- 2-bit saturating counter
Summary

- Make sure you understand the impact of different pipeline stages
  - How many delay slots for a branch? Why?
  - How many stalls needed for a RAW hazard? Why?
  - Why does it make a difference if a register read obtains the register value that is also written?

- Branch prediction
  - Can reduce number of required branch delay slots
  - Only requires a small amount of hardware
  - Useful for high performance

- Next time: considering processor performance